INTERNATIONAL JOURNAL OF MICROSIMULATION (2010) 3(2) 80-89

Techniques for Managing Changes to Existing Simulation Models

Sally Edwards
Department for Work and Pensions (DWP), Model Development Unit, Strategy Directorate, 6th Floor, Caxton House, Tothill St, London, SW1A 9NA, UK; email: Sally.Edwards@dwp.gsi.gov.uk

ABSTRACT: In This paper provides guidance for managing the enhancement and maintenance of an existing microsimulation model. It focuses on techniques and practices that have been developed to maintain Pensim2; the Department for Work and Pensions (DWP) dynamic microsimulation model that simulates state and private pension entitlement. This guidance is appropriate for any type of model. Standard project management techniques and bespoke simulation model procedures are explained. This guidance is aimed at new modellers and project managers and is particularly relevant for models that are supported by a team, rather than an individual.

Models are generally used for many years after they are launched, so it is essential that strong maintenance procedures and project governance structures are put in place. Specifically, this paper includes guidance and examples of structured Change Control processes that are used to manage enhancements and fixes to the model. The Change Control process includes: defining requirements, estimation, design, code reviews, testing/validation, managing multiple modellers changing the same programs and the use of development checklists. Structured Release management guidance is also included in this paper

1. INTRODUCTION

Typically dynamic microsimulation models are developed by Economists and Statisticians, who have an extensive knowledge of econometrics and the economic theory that underpins their models. Often modellers do not have a formal IT development background or training, so non-standard development and maintenance procedures are often used.

The purpose of this paper is to provide clear guidance in how to use standard techniques to manage the maintenance of existing dynamic microsimulation models, after the first version of the model has been released. This guidance is not limited to microsimulation models, but could be applied to any computer system, regardless of size, number of developers or users. Most of the recommendations documented in this paper are standard practices in use throughout the IT industry, although the examples have been tailored to suit the maintenance of a microsimulation model.

Microsimulation models typically take between one and five years to develop, depending on the level of complexity of the events being simulated. At the end of the Development Stage, the model is released to the organisation. Shortly prior to release, the model is thoroughly validated to ensure that the results can be used with confidence. Model users analyse the model outputs in order to assess potential and future economic policies. After the model has been initially released, the model moves into the Maintenance Stage. During this stage, a model is improved and enhanced, with new releases of the model being implemented on a regular basis, with rigorous validation taking place prior to each release. Usually a new release will contain a package of changes. Models need to be continuously enhanced and maintained otherwise they quickly become out-of-date and unusable.

Microsimulation models are based on a number of assumptions about the future which are obtained from external sources, e.g. inflation rates from the Government Treasury department, Demographic rates from the Government Statistics department. These assumptions are revised periodically and the models that use them must be modified to reflect these changes in order for the results of the models to remain credible. Enhancements are also required as a result of changes to legislation that must be reflected in the model.

It is essential that models are maintained and documented in a structured manner, so that the differences in outcomes from a new version of the model compared to the outcomes from a previous version can be explained and validated. Also, when questions arise about how a particular change has been incorporated into the model, it is essential that the associated documentation is clear and easy to locate. Documentation detailing how to change assumptions in the model is also essential as they need to be modified on a periodic basis.

Typically, there are more changes required than there are resources to code them, so some changes requested will inevitably be delayed or dropped. It is important that all the stakeholders, i.e. model users, senior managers and developers, have an opportunity to be involved in prioritising the requests and that a suitable forum is in place for this to occur. If stakeholders don't gain a sense of ownership and control over the development of the model, there is the risk that they may lose confidence in the model outcomes and the model will fall into disuse. This requires the change process to be well structured and accessible to all stakeholders.

This paper aims to provide simple, easy to follow

guidance. The techniques described are straight-forward and commonly used, but it took the Pensim2 team some time before these maintenance procedures were fully developed and utilised. It is hoped that these techniques may be useful to other modellers.

2. CONTEXT

2.1. Models at DWP

The dynamic micro-simulation models used at the DWP are based on the standard architecture, Genesis. This architecture enables each model to be coded in standard Excel spreadsheets that are easy to develop, understand and modify. These sheets define the parameters, the filters, the regressions and probability matrices used in the model. The Genesis engine generates the simulation model code from the Excel sheets and runs the model for the required simulation period.

The dynamic micro-simulation models generated using the Genesis modelling tool and used at the DWP are: Pensim2, which models private pension and state pension income to 2100; Inform, an Integrated Forecasting model for working age benefit claimants to 2020; a Suite of Benefit Forecasting models for short/medium term forecasts; and the Labour Force Survey (LFS) Model of employment & demographics to 2020.

2.2. Developing the Maintenance Procedures

A number of factors were taken into consideration when the DWP modelling team developed the procedures that are documented in this paper. These factors have caused problems in the past for the team, so the techniques that have been developed attempt to alleviate the risks and create an easy environment in which modellers can work.

Several modellers work on the same model at the same time, thereby creating the possibility that new code can be accidentally over-written if the development is not carefully managed. The modelling unit has a large staff turnover, with junior economists moving to a new post each year, so it is essential that the models can be quickly and easily understood by new team members. Each model has a range of stakeholders from across the Department for Work and Pensions. Some models also have stakeholders based in other government departments. These stakeholders are predominantly policy analysts and benefit forecasters.

The models are intended for long term use (15 years+) and consequently most of the people who use or maintain the models were not involved during the initial development stage. Hence the code and documentation must be as clear as possible. If the models were not structured in a manner that is easy for a new modeller to understand then the models would quickly fall into disuse and become un-maintainable.

Although these factors are not common to all modelling units, the general guidelines are particularly applicable if a model is run or maintained by more than one person, or if the model will be handed over to someone else in the future. Even when a model is maintained by one modeller, these guidelines may still be useful to follow – as they do not rely solely on a good memory.

3. DOCUMENTATION

This section outlines the documentation that the DWP modelling team have developed and maintain for the Pensim2 model and the Genesis modelling tool. This level of documentation is important for the modelling team, particularly the department has a large turnover of users and modellers, who need to gain a quick understanding of the model. Not all models will need the level of documentation listed here, but the documents considered essential for all models have been marked.

Some of the documentation that was produced during the development stage of the project was not considered relevant to keep up-to-date during the maintenance stage. This included detailed technical design specifications and the justification for design & development decisions.

The analytical papers supporting each module within the model have been filed and are accessible to all users.

The following key documentation for the Genesis modelling tool is kept up-to-date and reviewed and revised with each release of the modelling tool: Training materials, User Guide and Problem Resolution Guidance.

For the Pensim2 model, the following documents are kept up-to-date and revised with each release: How to Run the Model guidance; Assumptions audit; Release Document; also all changes to the model are documented in detail, as explained in the Change Control section below.

Much of the documentation was produced after the first micro-simulation model was released. There was a consolidation period, during which time the essential documents were produced and the maintenance procedures were put in place. These documents and processes have been invaluable, as there are a large number of users who access and run the models themselves and also a high turnover of modelling and user staff. Without the supporting documentation, extensive usage of the models would not have been viable.

3.1. Training materials

A short training course is run on a regular basis for new users. The training course materials include a Powerpoint presentation, sample models and a self-study pack, which explains how to

develop and run a Genesis model. The training material is updated with each new release of the Genesis tool.

3.2. User Guide (Essential)

An extensive User Guide provides a detailed explanation of the Genesis modelling tool, particularly explaining how the standard spreadsheets are defined and the options that can be used. It is a comprehensive reference manual and is used by modellers to dip into when questions occur about the syntax of the templates spreadsheets.

3.3. Problem Solving Guidance

Users have full access to the dynamic micro-simulation model code at DWP. They are provided with a training course when they begin to use the models. The users can modify the model parameters and the code, and run their own versions of the models to enable them to try out various policy scenarios.

The flexibility this provides for the users consequently has a support overhead for the modelling team, as the users sometimes inadvertently introduce errors into their own versions of the models.

A Problem Solving Process and a Problem Solving Guide are available for the model users. The process lists the steps to be followed take when encountered with a problem – one of these steps is a referral to the Problem Resolution Guide. The Problem Resolution Guide lists all the errors that are issued by the Genesis code generator and provides an explanation of the likely cause and suggestions to fix the code.

We have found the Problem Solving Guide to be particularly useful, as errors tend to re-occur, but their resolution is often forgotten. This Guide has been built up over the years as is editable by all users.

3.4. How to Run the Model (Essential)

A clear, concise document explaining how to run the model is essential for all models. This is written for a new user with no previous experience of simulation modelling.

3.5. Assumptions Audit (Essential)

An Assumptions Audit log containing the changes to the assumptions in the model is an important document to maintain. Assumptions are often questioned, so a clear explanation of where they have come from and when they were last updated is vitally important.

3.6. Release documentation (Essential)

When each Release is implemented, a Release Note is issued explaining how the specific changes cause the differences in the outcomes and how the changes interact to give overall differences. The Release Note is aimed at users and does not include any technical detail, but it includes references the change request and problem logs that have been included in the release. If desired, the user can review the change in more detail via the Change Request (CR) or Problem Log (PL) reference number.

4. MAINTENANCE PROTOCOLS USED AT DWP

4.1. Overview

Processes

The modellers use a set of key processes during the maintenance of a model. Each process is documented with an overview diagram and a checklist. The processes followed are: Change Management, Problem Management, Release Management, Software Configuration Management. Examples are provided below for each of these processes.

4.2. Change Management and Problem Management

The Change Management and Problem Management processes are very similar and this guidance applies to both changes and problems. It is recommended that changes and problems are recorded separately as problems are generally considered high priority and they usually need to be fixed in the next release of a model (see Figure 1 – Change Request / Problem Log process).

Every change / problem fix goes through a formal Change Request / Problem Log procedure. The term "change" in this guidance is used to represent both changes to the original requirements and to "problems", i.e. errors in the code that need to be fixed.

A change to the model may be generated by a request from any user of the model, a regular scheduled change to specific assumptions or a request by a modeller to tidy or improve the model without affecting the outputs.

A Change Register / Problem Log Register is maintained listing all changes carried out to the model, including those that don't affect the results or are considered trivial. Each Change Request (CR) / Problem Log (PL) is assigned a reference number (see Figure 2 – Change Control Register).

All changes are formally recorded. A CR or PL form is completed before the model code is amended. The change requirements or problem details are clearly documented in the CR / PL form – this may be carried out by model user or a member of the modelling team. Any background material that is relevant is also included and occasionally strings of emails are added to the CR / PL form. Documenting the requirements and supporting evidence is particularly important, especially when the model assumptions are being modified, as these are often questioned at a later date (see Figure 3 – Change Control Form).

The impact of the change is assessed, an estimate is produced and the modelling team assess whether or not the change is viable and sensible. Each model has a User Group, which consists of individuals representing the users and the model developers. The User Group assess all the changes put forward and determine whether or not to include the change in the model, based on the cost and the impact. The User Group also agree which release of the model is appropriate for the change.

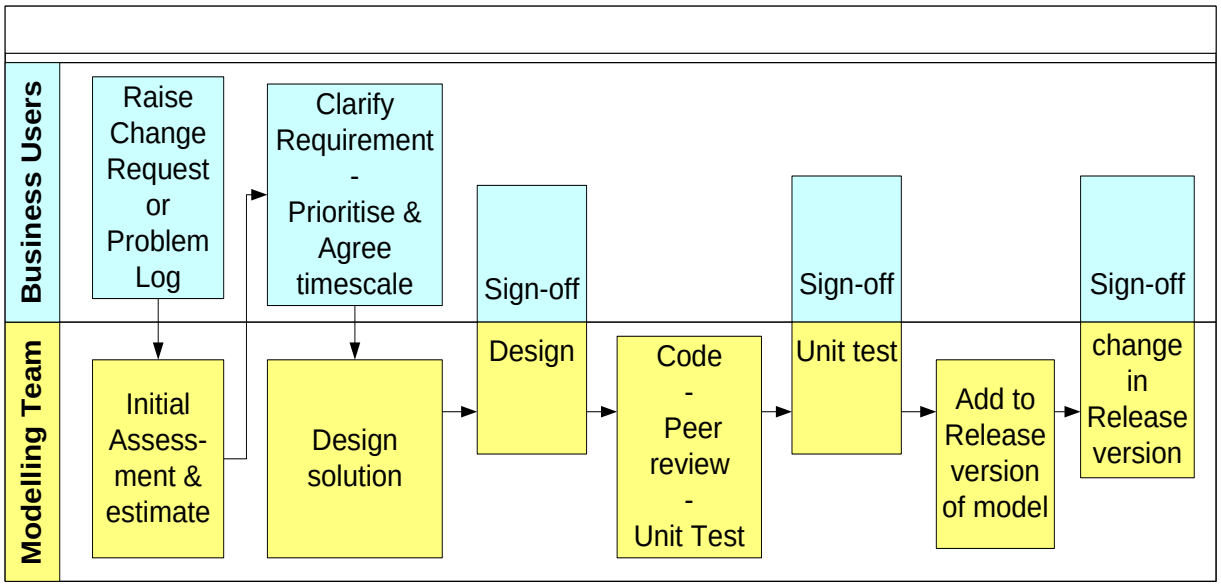
Where a large change is taking place, the design is documented and reviewed by the users before it is included in the model. This is a valuable stage that ensures that the requirement has been accurately understood and this enables the users to gain a better knowledge of how the model is structured. The design stage is signed-off before the model code is changed.

Complex code changes are peer reviewed by another member of the modelling team. All code changes that are associated with the change are commented with a reference to the CR / PL number.

Each change is added to the Released "live" version of the model one at a time. This enables the user who has requested the specific change to check that the outputs are appropriate and to sign-off the change for inclusion in the next Release. Different requirements are not bundled up into a single change request. By separating the individual changes, it is easy for the impact of each specific change to be clearly identified and it also enables a specific change to be easily removed from a release if necessary.

Change Request / Problem Log Process

Figure 1 Change Management and Problem Management Process, as detailed in section 4.2



Change Control Register – sample

Figure 2 Sample Change Control Register, as detailed in section 4.2.

CR No.	Summary of proposed change	Raised by	Date raised	Priority (H/M/L)	Who?	Estimate	Status	Release
CR0052	Revise retirement age mortality equations	State Pensions team	10/11/09	High	Steve	2 months	in progress	Planned for release V09
CR0053	Remove YD table and all references	Modelling team	05/02/09	Medium	John	1 week	released	Included in release V08
CR0054	Revise Fertility equations	Demography Unit	11/09/09	Low	John	4 weeks	on hold	Post release V09
CR0055	Add new Migration module	Demography Unit	11/09/00	Medium	Cathal	6 months	in progress	Planned for release V09
CR0056	Revise Private Pension membership equations	Private Pensions	11/09/00	Medium	Sally	6 weeks	on hold	Post V09
CR0057	Revise disability module for SDP	Disability team	13/09/09	High	Thomas	2 months	released	Included in release V08

Note: all changes to the model are included in the register, including those that do not affect the outputs.

Change Control Form - sample

Figure 3 Sample Change Request, as detailed in section 4.2.

Change Control Form	
CR Number	CRnnnn
1. Project Name	
2. Raised by (name & contact details)	
3. Date Raised	
4. Details of Change (including reasons for change & benefits where applicable)	
5. Importance	**Reason**
❑ Mandatory ❑ Essential ❑ Highly Desirable ❑ Desirable	
6. Change Sponsor	
7. Summary of Impacts	
8. Estimate	
9. Decision	**Reason**
❑ Accepted ❑ Rejected ❑ Escalated ❑ Deferred	

Note: a form is completed for every change, including those that do not affect the outputs.

4.3. Release Management

New releases of models are carefully controlled and thoroughly validated before they are released to the user community. Each model has a new Release implemented approximately every 6 to 9 months. If a major problem occurs and needs to be fixed before a scheduled release is implemented, then a sub-release is considered, but these are rare and most problem are fixed in the next scheduled release.

Each Release includes one or two large components, e.g. new Base Data, new Migration module, new State Pension policy, together with a number of smaller changes and problem fixes. Usually between 20 and 30 changes are implemented in each scheduled release.

The provisional timing and content of each Release are agreed with the Model User Group several months before the release date, with additional changes accepted up until the cut-off date, three weeks before the Release date.

Each Release is built and signed off in a step-by-step process, with each change added to the model individually, as the new release is built (see Figure 4 – Release Control Log). The impact of each change is reviewed & signed off by the user who has requested the change and also by any other users who are interested in the difference in outputs caused by change. By implementing the changes one at a time, the impact of each change can be fully understood and agreed. Although this is a time consuming process, it gives the modellers and users confidence in the results and removes the stress, confusion and risk from the release process. The final stage in each release is to remove redundant code and variables that are no longer required.

Prior to adopting the step-by-step approach it was very difficult to explain the differences in outputs between different releases, particularly when changes conflicted with each other. It was also easier to inadvertently slip in an error to the model when we attempted to implement multiple changes at the same time.

All previous versions of the model remain available for access, if required. A clear audit trail is maintained detailing the new functionality included in each release. This is documented in a Release Note that is issued with each release of the model.

Maintaining a checklist of the tasks that need to be carried out for a Release can be particularly useful (see Figure 5 – Release Checklist).

Release Control Log - sample

Figure 4 Sample Release Control Log, showing how each change is added to the model code and signed off one at a time

V09_00 Test Versions						
Test version	**Added to version**	**Change**	**Run to**	**Due**	**Sign-off**	**Status**
V09_A	**V08**	+ PL0178 Correct Accumulation errors	2050	Dec-08	Model developers only	signed-off
V09_B	**V09_A**	+ CR0101 Revised Private Pension Assumptions	2060	Jan-08	Private Pension Users	signed-off
V09_C	**V09_B**	+ CR0122 new Housing Module	2100	Jan-09	Housing Users	signed-off
V09_D	**V09_C**	+ CR0134 Revised Economic assumptions	2100	Feb-09	All Users	with users awaiting sign-off
V09_E to V09_S						
V09_T (Final V09)	**V09_S**	+ CR0144 remove redundant code	2100	Jun-09	Model developers only	

Release Checklist – sample

Figure 5 Sample Release Checklist

Release: V08		
Release Date: end of June 2008		
No	**Step**	**who**
1	Agree Release date. Major releases will be planned in a long term release schedule. Minor releases may be required to carry out critical fixes, key assumptions changes or systems changes	User Group
2	Agree Release content and prioritise changes	User Group
3	Agree priority for each element of the release with the user group	User Group
4	Agree Release Freeze date, generally 3 weeks before Release date, after which time new changes are not accepted	User Group
5	Determine order in which the changes are going to be incorporated	Modelling team
6	Maintain the Change Control Matrix throughout Release development	Release manager
7	Agree key criteria for Test Versions, e.g. sign-off responsibility, timing	Release manager
8	Ensure development tasks have been completed for each component	Release manager
9	Build each test version	Model developer responsible for change
10	Request sign-off for specific test version	Model developer responsible for change
11	For changes that are not signed-off, determine whether the change should be removed from the release. Liaise with and notify user group members and development team	Release manager
12	When a test version is signed off, update the Test Version log and Change Control Matrix (shade green to show change included)	Release manager
13	When a change is removed from a release, update the Test Version log and Change Control Matrix (shade red to show change removed)	Release manager
14	Remove redundant code	Modelling team
15	Produce/maintain documentation, including: Release Note, How to Run the Model Guidance, Assumptions Audit	Release manager
16	Create a new version of the How to Run the Model guidance	Modelling team
17	Has the base data changed? If so, create new base datasets	Modelling team
18	Copy the final test version to the Released area	Release manager
19	Notify all users that a new release of the model is available	Release manager
21	Carry out Post Implementation Review, if appropriate	Modelling team & User Group

4.4. Software Configuration Management

The term software configuration management is a general term that is used to describe the control and maintenance of software. This is essential for a number of reasons. A modeller may need to access or re-instate a previous version of the code and an audit trail of changes must be maintained. It is also important if multiple people work on the same model at the same time. It can be difficult to keep track of changes to the code and easy for one person's change to inadvertently over-write another change that is being implemented by someone else in the same release.

Most organisations use standard source code control software, and it is recommended that this software is used if it is available. If not, then a Change Control Matrix (Figure 6 – Change Control Matrix) can be used to track the changes to

specific modules within a model. The column headings show the test version reference number and the individual cells indicate which modules have been changed for each test version. As a release is built, modellers can identify whether a module they are changing has been altered by another member of the team, as different sets of changes may need to be integrated. In the example, columns are colour coded to indicate when a test version has been signed-off (green) or awaiting sign-off (yellow). Each change is applied to the previous signed-off version of the model, with the individual changes added one at a time.

Change Control Matrix - Sample

Figure 6 Sample spreadsheet used for Software Configuration Management

Change Matrix - V09	V09_00A	V09_00B	V09_00C	V09_00D	V09_00E	V09_00F	V09_00G
	Fix pension start date	Remove redundant tables	Hide historic rows	Alter Cont-ribution rates	New Housing module	Fix Pension age error	De miminis rule
CR/PL	**PL0181**	**CR0112**	**CR0074**	**CR0116**	**CR0179**	**PL0178**	**CR0127**
PrivPenInh	Mod						
TrivialCommutation	Mod						
TCSmallPots	Mod						
ClaimPension	Mod					Mod	
PersonJoinScheme	Mod		Mod			Mod	
PersPenMembership				Mod		Mod	
GrossAPAccrual		Mod	Mod				
Percentile					Del		
IncomePercentile					New		

5. PROJECT MANAGEMENT

5.1. Project Planning – High Level

High level project planning is recommended in order to provide a clear idea of the work required for each release of the model – this also enables the project manager to identify early when the release is slipping.

A project plan provides the teams involved in the release with an indication of when specific resources will be required during the release cycle. This is important when work is delegated to other teams, who may be needed during the user-test stage. Individuals can be allocated to tasks to aid planning where required.

MS Project or basic Gantt charts in Excel are the easiest tools to use to produce plans (Figure 7 – High Level Plan). It is usually unnecessary to plan the work at a very detailed level, as detailed plans get out of date quickly and need frequent re-work. If a release starts to slip, it's that the users are notified as soon as the problem is identified. They can then choose between keeping the original delivery date and reducing the scope of the release or allowing the release to be delayed.

High Level Plan - sample

Figure 7 Sample High Level Plan, showing anticipated involvement from user teams

	Feb	Mar	Apr	May	Jun
Planned Releases					
V08					
Modelling team					
CR0123 - Out Of Work States	TJ				
CR0126 - New Private Pension assumptions	TJ	TJ	TJ		
CR0126 - New Base Data	HR/RP	HR/RP	HR/RP	HR/RP	HR/RP
CR0129 - Migrate Pensim2 to Genesis76		JA	JA		
CR0130 - Remove redundant code for V08				JA	
Release V08					SE
Demographic Team					
CR0096 - Revised mortality equations	IR				
CR0120 - Fertility		SM	SM		
Pensions Business team					
Testing V08					
Forecasting Division team					
Testing V08					

5.2. Governance – User Groups and Steering Groups

Strong project Governance structures are essential for a successful model. It can take a while to develop the most appropriate governance structures and the membership of these groups and their roles should be regularly reviewed to ensure that they meet the needs of the customers appropriately.

For a large, high profile model, a Model Steering Group should be considered, consisting of senior stakeholders. This group will not meet often (maybe quarterly or semi-annually). Their purpose is to provide clear direction and prioritisation of the major future developments.

Each model needs an enthusiastic User Group, which should meet frequently (e.g. once every 3 weeks). The User Group should consist of a representative from each main group of the model users. The members of the User Group will propose potential changes and agree the detailed requirements and priority of every change added to the model. The members of the group are also responsible for reviewing the outputs and signing off the changes, before they are released.

The Model User Group plays a key role in ensuring that the model is enhanced in the way that will make best use of the limited resources available. A word of warning – it can be difficult to establish an enthusiastic User Group, but it is worth persevering, as they provide valuable input to your model. During the first year or so after the Pensim2 model was released we had only 1 or 2 regular members of our User Group, but over time the popularity of the group (and the model) has increased and the group now has up to 10 users who regularly attend the User Group session.

6. DIAGNOSTICS AND SUMMARY TOOL

6.1. Diagnostics Macro Tool

The dynamic micro-simulation model code used at DWP is predominantly written using standard Excel sheets that are then used to generate the model. These sheets must follow a precise template and the cross-references must be in place.

A Diagnostic macro was produced in VBA to check that the syntax of the code is valid and to ensure that all the components of the code are accurate and link together correctly, e.g. if a variable is used in the code, then it must previously have been defined in the Data Dictionary.

The Diagnostic macro is run before the model is simulated and it produces a list of errors, if there are any. These must be removed before the simulation takes place.

The Diagnostics macro has proved invaluable, as it saves a considerable amount of time during testing. I strongly recommend that modellers consider producing a Diagnostics macro to valid the syntax of the model code, if it is appropriate for their model.

6.2. Summary Macro Tool

A Summary macro has been developed in VBA, which reads through the model spreadsheets and produces a standard document showing where each variable is assigned as an output variable and also where each variable is used as an explanatory variable or within a selection filter.

This is also a particularly useful tool, which is used to help the modellers find their way around a model and to identify the relationships between variables. This documentation is automatically generated by the macro and is produced whenever a new release of the model is implemented.

7. PERSONAL RECOMMENDATIONS

This section of the paper contains my personal recommendations for techniques that should be considered when maintaining models. By adhering to this guidance, you will help to make your model easier to maintain and hence lengthen its life. When any system becomes overly complex, it gradually reaches the point where no-one can understand it, except its author, and hence it is no longer usable by anyone else. Well written and well maintained models can be picked up and understood by another modeller. It is recommended that this is a key aim when a model is built.

7.1. Tidying Up Code

Don't ever tidy up code as part of another change. Always create a separate change request, carry out the changes to tidy up the code and then compare the outputs before and after the changes have been applied – they should be identical.
Remove redundant code and variables as the last step before a release is implemented. Again, carry out a comparison test on the outputs before and after the changes to ensure that the results are not affected. Leaving redundant code in a model causes confusion for future modellers. It is considerably quicker to remove redundant code when it becomes redundant rather than wait – future modellers will not know whether it is needed or not.

7.2. Keep it Simple – Avoid Complex Code

Write code that is easy-to-read – always consider the next person who will be attempting to understand your code, as you may not be around to explain. Considerably more time is spent reading code than writing it, so it needs to be easy to understand.

Add lots of comments, using colours for clarity where possible. If you have spent some time working out what a section of code does, then add a detailed comment with an explanation. Indent your code to help readability. Write code in small neat modules that can be more easily understood.

If your code is too difficult to understand, it won't survive.

7.3. Keep a tidy Directory structure

Although this sounds trivial – it is an important point, as new team members need to be able to find documents easily.

It is recommended that the top directory level is restricted to 10 sub-directories. Consider numbering them, so they are ordered as you want to see them

Make your directory structure intuitive – this is easier said than done. The aim is for someone to be able to easily find what they are looking for.

7.4. Maintaining a Model with no existing Documentation

If you are maintaining a model that does not have clear documentation, then as a minimum starting point, it is suggested that you consider the following:-
Produce a High level presentation explaining the model, including diagrams; Write a guidance document explaining how to run the model; Add comments to the code when changing the model; Maintain an Assumptions audit log, when changing the underlying model assumptions; Set up a Change Control system; Produce a Release Checklist and issue a Release Note with each new version of the model.

Suggested Further Reading:

BUSINESS ANALYSIS by Don Yeates, Debra Paul, Tony Jenkins, and Keith Hindle

ESSENTIAL SOFTWARE ARCHITECTURE by Ian Gorton

PRINCE2 EDITION 2009: A Pocket Guide by B; Seegers, R Hedeman